Pixel Programming Language

Final Project - Team 16

Emerging Languages and Programming Paradigms

Team Members

Group 16

- Swapnil Mukeshbhai Chadotra
- Sudheer Reddy Kunduru
- Jashmin Mineshkumar Patel
- Jay Yogeshbhai Patel
- Manan Hasmukhkumar Patel

Contents

Introduction

Pixel is an efficient programming language that was developed as part of SER 502 Emerging Languages and Programming Paradigms course.

Tools and Components

Tools

Lexical Analysis - Python (tokenize, tkinter)

Language Grammar - Definite Clause Grammar

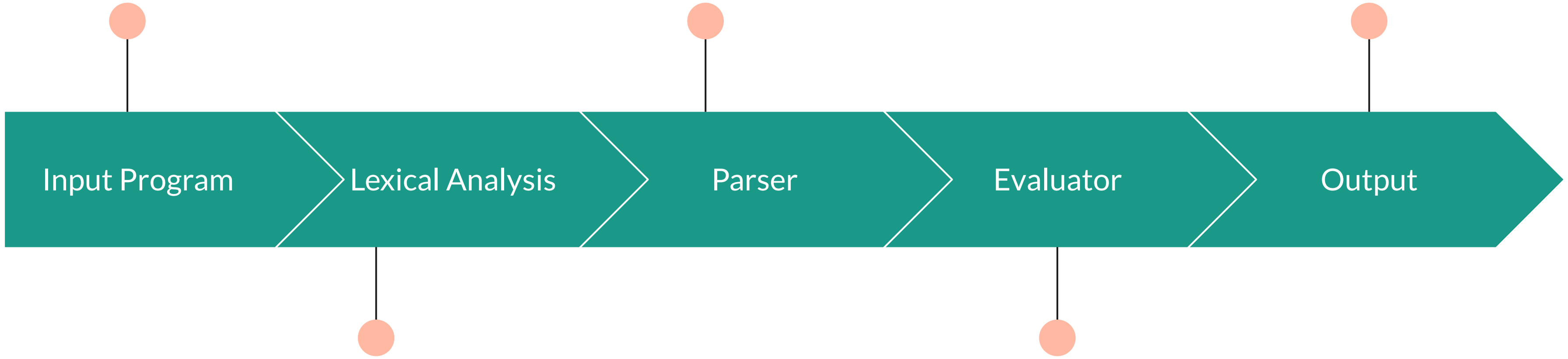
Parse tree generation - SWI Prolog

Semantic Evaluation - SWI Prolog

The user creates a program using the provided features and syntactic sugar

The parser checks the grammatical correctness of the program and generates the parse tree

The program's output will be displayed to the user



The lexical analyzer breaks down the program into a list of tokens

Evaluator maintains the runtime environment and executes instructions

Lexical Analysis

The Lexical Analyzer is responsible for breaking down the source code into tokens.

The Lexical Analyzer has been implemented in Python using the tokenize and tkinter libraries.

The Lexical Analyzer eliminates unwanted white spaces and new line characters and converts the remaining code into meaningful tokens.

Parser

- The tokens generated by the lexical analyzer will be used for parse tree generation by the parser.
- The parser has been developed using a Definite Clause Grammar that reflects the language’s grammar design.
- The parser performs grammar checks on the user program to see if the program is syntactically correct or not.
- The parse tree will be generated upon determining that the program is grammatically correct.
- The parser has been implemented using SWI Prolog.

Evaluator / Run time Manager

- The evaluator is responsible for checking the semantics of the program.
- The evaluator adds the declared variables to the runtime environment and maintains/updates them.
- The evaluator performs semantic checks to avoid incorrect operations such as updation of constant entities, and updation of a variable to an incompatible type etc.
- The evaluator raises exceptions and displays error messages for any faults within the programs.
- The evaluator displays the output of the programs to the user.
- Evaluator has been implemented using SWI Prolog.

Language Design and Features

Variables and Constants

Three supported datatypes:

- Integer
- Boolean
- String

Two types of entities:

- Variable
- Constant

Identifiers for all the variables and constants are single letters

Keywords used:

- **var** - for variable declaration
- **const** - for constant declaration
- **int** - declaration of an integer value
- **bool** - declaration of a boolean value
- **str** - declaration of a string value
- **true** - Boolean true value
- **false** - Boolean false value

Variables and Constants (Continued)

- Strings can be surrounded with either single quotes or double quotes
- A boolean entity assumes either true or false value.
- All declarations and assignment statements end with a semicolon (“;”)
- The value of a variable entity can be changed but not that of a constant entity
- Pixel is statically typed, i.e., a variable of a type can’t be assigned values of a different type

Declaration examples:

- **var int x = 10;**
- **var bool b = true;**
- **var str s = “Hello”;**

Assignment examples:

- **x = 100;**
- **s = “true”;**
- **x = 2 * 3 - 5 + 6 / 2;**
- **b = (3 * 8 - 2) < (6 * 3);**

Operators

- Arithmetic Operators:

+	**Addition**
-	**Subtraction**
*	**Multiplication**
/	**Division**

- Assignment Operator **=**

- Boolean Operators:

and	**Logical AND**
or	**Logical OR**
not, !	**Logical NOT**

- Ternary Operator:

  <condition> ? <command1> : <command2>

Relational Operators

>	Greater Than
<	Less Than
>=	Greater Than or Equal to
<=	Less Than or Equal to
==	Equal To

Usage:

var bool x = 4 < 5;

var bool y = (3 * 8 + 2) <= 6;

Program Structure

Every program in Pixel begins with main and an opening curly bracket (“ { “) followed by a series of declarations, commands and control structures.

The program ends with a closing curly bracket (“}“).

Every Pixel program must have the file extension “.pixel”.

Except control structures, all the statements within the program should end with semicolon (;).

Example Program:

```
main {

var int x = 30;

const bool a = true;

print(“Hello”);

}
```

Simple Commands

- print() - Function used for displaying the output
- print() takes a single input which can be a value or a variable of any type.
- Assignment statements are of the form LHS = RHS;
- The LHS for assignment consists of the identifier for an already declared variable and not a constant.
- The RHS consists of a value or an expression.

Examples:

```
print( 5 );

print( “Hello” );

var int x = 20 *60 - 30 / 2;

print( x );

x = 20;

x = 20 /2 + 30;
```

Special Operators for integers

+= - Syntactic sugar for addition and updation

-= - Syntactic sugar for subtraction and updation

***=** - Syntactic sugar for multiplication and updation

/= - Syntactic sugar for division and updation

++ - Increment by 1

-- - Decrement by 1

Examples:

- **a += 2; - equivalent to a = a + 2;**
- **a ++; - equivalent to a = a + 1;**
- **var int b = a ++; - equivalent to var int b = a; a = a + 1;**

Control Structures

The supported control structures are:

- **if**
- **if - else**
- **if - elseif - else**
- **for**
- **while**

The body of all the control structures begins with “{“ and ends with “}” just like the main block.

Variables and constants defined within the control structures have local scope.

The keywords used within the control structures are:

- **if**
- **else**
- **else if**
- **for**
- **while**
- **in**
- **range**

Conditionals (if - else)

The if block begins with the “if” keyword followed a boolean condition within parenthesis and a series of statements within curly brackets.

The optional else block begins with “else” keyword and contains instructions within curly brackets.

The if block is executed if the boolean condition evaluates to true, otherwise the else block is executed.

Examples:

```
if ( 4 < 5 ) {

print( “Less than” );

}

else {

print ( “Greater than or equal”);

}
```

Conditionals (Else if ladder)

Between the if block and the else block, any number of else-if blocks can be optionally added.

The structure of an else-if block is similar to an if block but has the keywords “else” and “if” at the beginning.

Else if ladder is used for checking for a series of conditions where the first block whose condition evaluates to true is executed.

Example:

```
if ( a < b) {
print ( “a is less than b”);
} else if ( a == b) {
print ( “a is equal to b”);
} else {
print ( “a is greater than b”);
}
```

For Loop

Four flavors of for loop are supported:

```
for ( <declaration>;<condition>;<assignment>) { }

for ( <assignment>;<condition>;<assignment>) { }

for <variable> in range (<start>,<end>,<jump>) { }

for <variable> in range (end) { }
```

The first two loop flavors are similar to the regular for loops used in C / Java.

The third flavor initializes the variable to <start>, and increments/ decrements its value by <jump> after every iteration, the loop executes until the variable’s value reaches or exceeds <end>.

The last flavor is similar to the third flavor except that the <start> value is initialized to 0 and the <jump> value is initialized to 1.

For Loop Examples

```
for ( var int i = 0; i < 5; i ++ ) {
print( i );
}
var int x = 1;
for (x = 100; x < 600; x += 100) {
print( x );
}
```

```
for p in range ( 1, 10, 2 ) {
print( p );
}
for a in range( 5 ) {
print( a );
}
```

While Loop

The while loop has the following structure:

```
while ( <condition> ) {
<Statements>
}
```

The statements within the while block are executed iteratively until the <condition> evaluates to false.

Example:

```
var int a = 0;
while ( a <= 3) {
print( a );
a ++;
}
```

Execution Steps

Unzip the Folder from github named SER502-Spring2023-Team16

Execute lexical_analysis.py

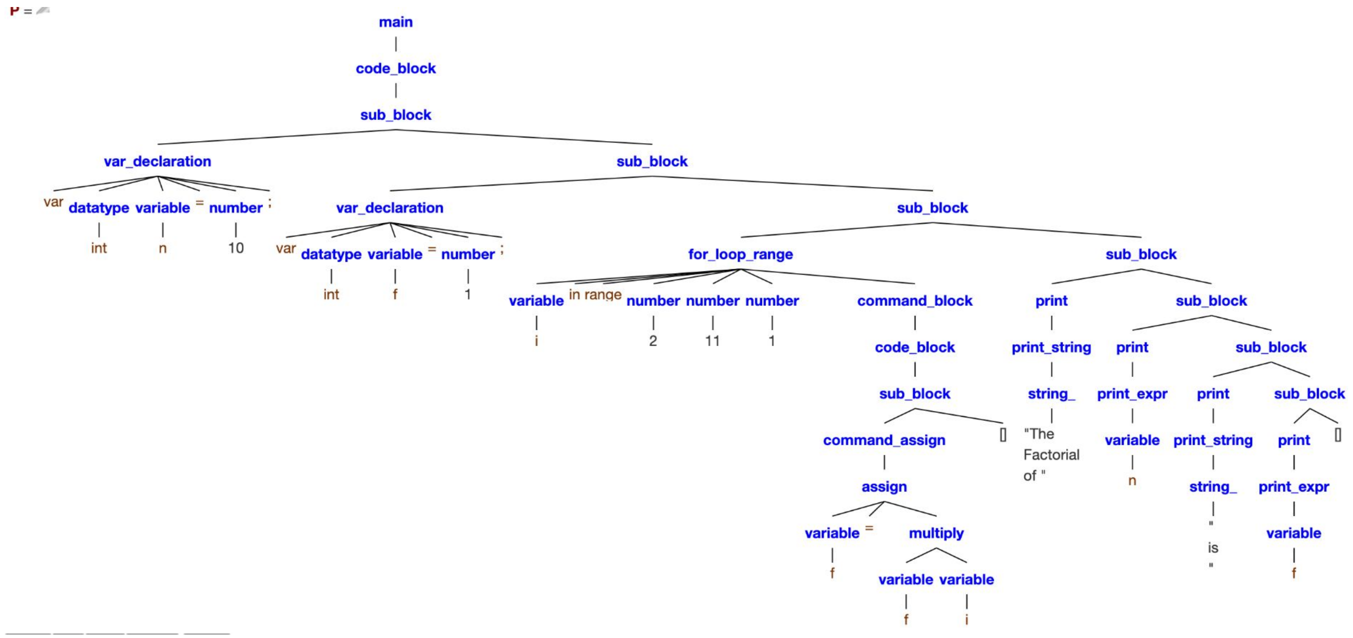
A file dialog will be displayed for selecting the program to execute

Then, select a file with .pixel extension and click on “Open”.

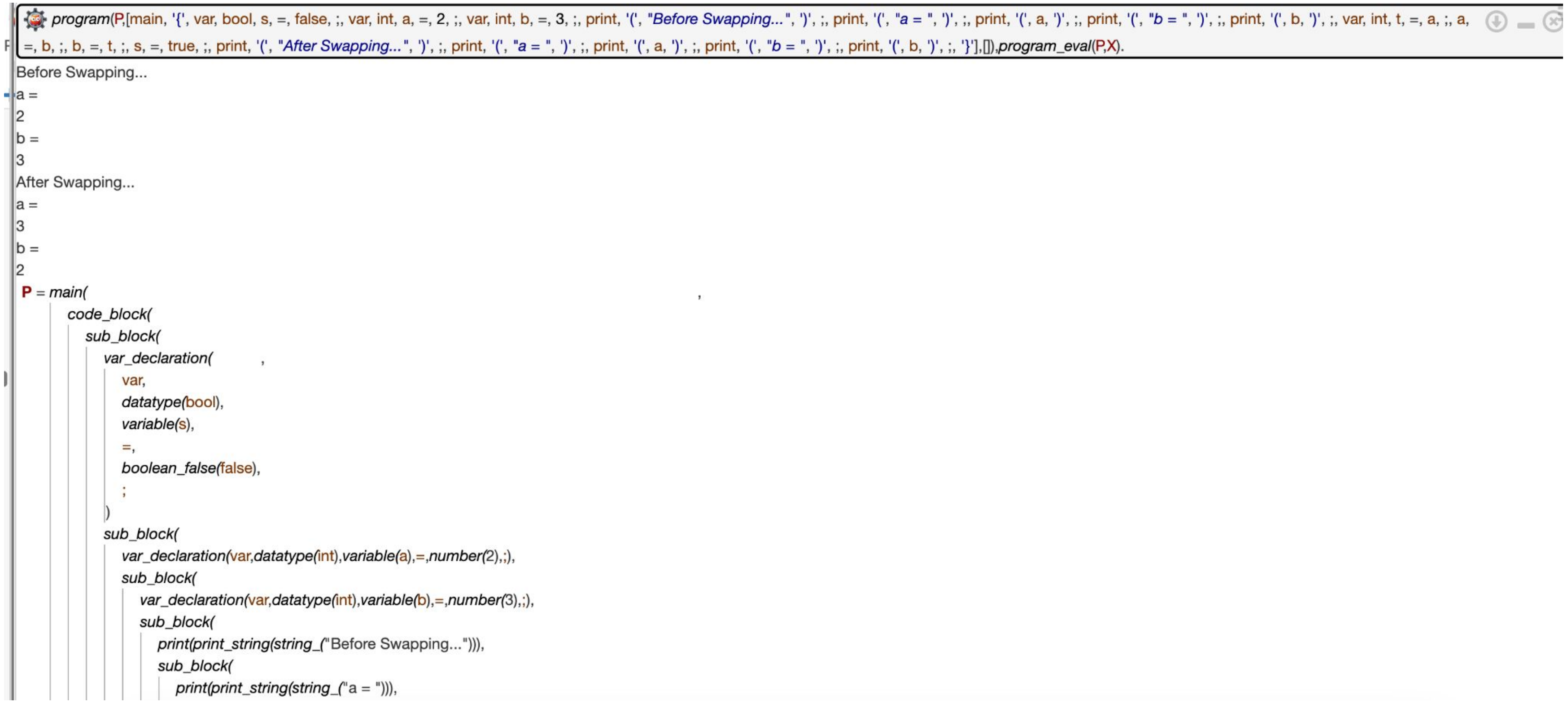
The corresponding parse tree for the program and its output will be displayed in the console.

Parse tree output

Parse tree for program to compute factorial of a number



Sample Execution Output



```
program(P,[main, '{', var, bool, s, =, false, ;, var, int, a, =, 2, ;, var, int, b, =, 3, ;, print, '(', "Before Swapping...", ')', ;, print, '(', "a = ", ')', ;, print, '(', a, ')', ;, print, '(', "b = ", ')', ;, print, '(', b, ')', ;, var, int, t, =, a, ;, a,
=, b, ;, b, =, t, ;, s, =, true, ;, print, '(', "After Swapping...", ')', ;, print, '(', "a = ", ')', ;, print, '(', a, ')', ;, print, '(', "b = ", ')', ;, print, '(', b, ')', ;, '}'],[]),program_eval(P,X).
Before Swapping...
a =
2
b =
3
After Swapping...
a =
3
b =
2
P = main(
        code_block(
            sub_block(
                var_declaration(
                    var,
                    datatype(bool),
                    variable(s),
                    =,
                    boolean_false(false),
                    ;
                )
                sub_block(
                    var_declaration(var,datatype(int),variable(a),=,number(2),;),
                    sub_block(
                        var_declaration(var,datatype(int),variable(b),=,number(3),;),
                        sub_block(
                            print(print_string(string_("Before Swapping..."))),
                            sub_block(
                                print(print_string(string_("a = "))),
```

Thank You